React

- Vorteile Oberflächen-Entwicklung per DOM-API
- Erweiterung Server-lastiger Websites mit Client-seitige Interaktion
- Webseiten müssen keine Abhängigkeiten nachladen und lassen sich früher vom Anwender bedienen (Time-to-Interactive)

Konzept hat ausgedient, wenn große Benutzeroberflächen und viel Client-seitiger Benutzerinteraktion, im Extremfall sogar Single-Page-Application (Single-Page-Webanwendung oder Einzelseiten-Webanwendung = Webanwendung, die aus einem einzigen HTML-Dokument besteht und deren Inhalte dynamisch nachgeladen werden.)

hier wäre viel Aufwand zu betreiben, DOM-Befehle und Abfragen, korrektes Timing von Events und Render-Vorgängen... braucht anderes Konzept

React = deklarativ

grundsätzlicher Ansatz von React:

müssen nicht mehr beschreiben, wie sich Darstellung in andere ändert, sondern
wir beschreiben (deklarieren) die Oberflächen anhand von Funktionen, die HTML - ähnlich wie Templates - enthalten

Änderungen an variablen Teilen der Funktion veranlassen React dazu, selbst DOM Befehle zu ermitteln und auszuführen

React übernimmt eigentliche Arbeit - wir sagen nur noch was wir haben möchten - React setzt um

React = Komponenten-basiert

React bietet Möglichkeit, Komponenten zu entwickeln

Komponenten lassen sich zentral warten und an unterschiedlichen Stellen wiederverwenden

React setzt dafür auf HTML-Modell

Komponente stellt sich als Custom-HTML-Tag dar und kann genauso einfach verwendet werden

React = Technologie-agnostisch

React versucht nicht alle Probleme auf einmal zu lösen

konzentriert sich primär auf die GUI

dadurch möglich, React einfach und beliebig mit anderen Frameworks zu kombinieren

React ist nicht nur auf den Browser beschränkt

in Verbindung mit NodeJS können wir Seiten schon auf Server rendern, um Perfomance zu verbessern oder suchmaschinenfreundliche Ergebnisseiten auszuliefern

Arbeitsweise

können mit ReactJS rasend schnell reagierende Webseiten entwickeln

echtes Browser DOM = komplexes Gebilde - bremst Performance stark aus

React kommt mit eigenem entschlackten DOM, das nur genau die Features hat, die React tatsächlich benötigt: das **Virtual DOM (kurz VDOM)**

Änderungen im VDOM zunächst einmal abstrakt und werden nicht automatisch sichtbar

2. Teil von React, das npm-Paket react-dom, bringt mächtigen Diff-Algorithmus mit
= Algorithmus, der virtuelles DOM mit dem echten vergleicht

Algorithmus analysiert Unterschiede, berechnet kleinst-möglichen Befehlssatz, der nötig, um echten DOM mit virtuellem zu synchronisieren und bringt realen DOM automatisch auf aktuellsten Stand

Algorithmus von React hervorragende Performance

Babel = Transpiler-Toolkit

ermöglicht z.B. immer neueste ECMAScript-Syntax auch auf älteren Browsern auszuführen - transpiliert dazu JavaScript auf ältere ECMAScript-Version (typischerweise 5.1) zurück

im React-Kontext ermöglicht Babel Einsatz von JSX, einer HTML-ähnlichen Syntax in JavaScript

NPX - Tool zum Ausführen von Node Paketen

NPX wird gebündelt mit NPM Version 5.2+ geliefert

NPM führt nicht einfach ein Paket aus - muss zuvor in package.json angeben werden

NPX = kleines Dienstprogramm, das intelligent genug, um richtige Anwendung auszuführen, wenn es aus Projekt heraus aufgerufen wird

können npx nutzen, um Node-Paket auszuführen, ohne es tatsächlich zu installieren (bzw. durch temporäres Installieren, je nach Betrachtungsweise)

JSX

bei JSX (JavaScript XML) handelt sich um JavaScript-Erweiterung, die erlaubt, HTML-ähnliche Syntax direkt in JavaScript zu verwenden

sieht aus, wie selbst geschriebener HTML-Tag <Komponente /> ist sogenanntes JSX-Element

React baut HTML-Syntax relativ genau nach (mit ein paar nötigen Abweichungen)

JSX sehr einfach mit JavaScript-Funktionen um eigene Tags (React-Komponenten) erweiterbar

wichtig, Elemente aus HTML immer klein zu schreiben
eigene Tags beginnen immer mit einem Großbuchstaben
ist notwendig, damit React zwischen Standardkomponenten und selbst definierten unterscheiden kann

Props

JSX-Tags können eigene Attribute besitzen (wie Attribute in HTML-Tags)

so können wir Komponenten variabel einsetzen

React-Komponente nimmt dafür Properties (kurz: props) als Funktionsargument entgegen

props = Objekt, das die Attribute aus dem JSX-Element als JavaScript-Properties enthält

haben dann Möglichkeit, innerhalb der Funktion auf props zuzugreifen (geschweiften Klammern {} dafür nötig)

wie bei Templatesprache lassen sich innerhalb der geschweiften Klammern beliebige JavaScript-Ausdrücke verwenden

JSX != HTML

wurde so entworfen, um HTML möglichst ähnlich zu sein

handelt sich in Wirklichkeit um JS-Funktionsaufrufe, die lediglich in HTML-artiger Markup-Syntax dargestellt werden

bsw. wird aus JSX: lorem ipsum dol dourm et.

JavaScript-Funktionsaufruf: _react.default.createElement("p", null, "lorem ipsum dol dourm et."));

Aufgabe von Babel, JSX wieder in JS-Funktionsaufrufe zu konvertieren

Parcel übernimmt Aufgabe, Babel anzusteuern und Resultate automatisch auszuliefern

um zu sehen, wie generierter JS-Code tatsächlich aussieht, Blick ins dist-Verzeichnis werfen - dort liegen sämtliche generierte Dateien

prop-Objekt erhält automatisch Eigenschaft children ({props.children}), wenn JSX Element Kindelemente besitzt

können jetzt beliebiges HTML-Markup innerhalb einer Komponente nutzen - damit deutlich ausgefeilteres Markup möglich

möglich, Daten oder Komponenten aus Dateien zu exportieren und dann in index.js zu importieren

um Elemente beim Rendern zu unterscheiden, wird von React Attribut key mit einem eindeutigen Wert benötigt

Schlüsselwort class in JavaScript reserviertes Wort - deswegen in JSX stattdessen className nutzen

Komponenten, State & Events

können an Komponenten Event-Handler integrieren, um Verhalten zu steuern

In React wird Event-Handler als Attribut auf entsprechendem Element angegeben: mit on und dem Eventnamen — also bsw. onBlur - grundsätzlich CamelCase verwenden

als Event-Handler lässt sich beliebige Funktion hinterlegen, die typischerweise innerhalb der Komponente definiert wird - da Komponente auch Funktion darstellt, müssen wir Funktionen verschachteln

State und Hooks

wenn React-Komponente einen State (deutsch: Status/Zustand) verwalten muss, müssen wir diesen übergeben

State in React = Objekt, das aktuelle Werte aller benötigten Daten der Komponente hält

Komponenten, die keine eigenen Werte speichern - sondern Werte über props erhalten = stateless

Komponente, die einen State besitzt = stateful

bis React-Version 16.7 mussten Stateful Components als Klassen implementiert werden

modernere Variante ist **Hook useState**

useState aus React-Bibliothek importieren

import { useState } from "react";

wenn importiert, können wir sie aufrufen und nutzen:

const [status, Setter] = useState(Initialwert);

- Parameter von useState stellt Initialzustand dar

- Rückgabewert von useState = Array mit 2 Elementen:

1. eigentlicher State
2. Funktion zum ändern des States (ein Setter)

Werte lassen sich per Destructuring einfach in zwei Konstanten speichern

unkontrollierte Komponente = Uncontrolled Component

kontrollierte Komponente = Controlled Component

Unterschied ist, dass wir volle Kontrolle behalten, können Prüfungen für Eingaben integrieren etc.

Hook: useRef

mit useRef erhalten wir Möglichkeit ein DOM-Element als Komponente zu referenzieren

React-Referenz ist innerhalb der Komponente einzigartig

für jede Instanz der Komponente legt React automatisch neue Referenz an

Hook wird mit: const name = useRef(null); aufgerufen

Name der Referenz = beliebig wählbar

Referenz wird mit Attribut ref={name} an Komponente übergeben

damit Möglichkeit, jederzeit über ReferenzObjekt auf das referenzierte Element zuzugreifen : name.current

Hook: useEffect

mit useEffect lassen sich beliebige Nebenwirkungen auslösen (engl. SideEffect)

Hook wird von React automatisch nach dem Rendern aufgerufen

Hook muss aus react-Libary importiert werden

bei Aufruf der Funktion useEffect wird gewünschter Effect als Callback-Funktion übergeben

Hooks Gesetze

1. Hooks nur auf der obersten Ebene aufrufen - d.h. direkt in Komponenten
   nicht in Schleifen, Bedingungen oder verschachtelten Funktionen aufrufen

2. Hooks nur von React-Funktionskomponenten aus aufrufen
   nicht in regulären JavaScript-Funktionen aufrufen

State nach oben ziehen

Sind bestrebt, immer alle relevanten Daten in der gleichen Komponente zu verorten

Kann aber auch Probleme bereiten

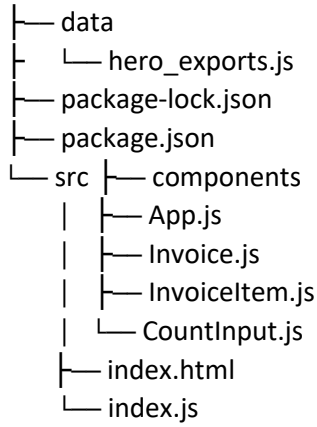
können state in übergeordneter Komponente aufrufen und Konstante sowie Funktion über props weiterreichen – so lassen sich diese in untergeordneten Komponenten nutzen

In größeren Projekten oft viele miteinander verzahnte Komponenten
kann Handhabung von veränderbarem State oft schwierig machen
genügt nicht mehr, State nach oben zu ziehen

hier helfen sogenannte Reducer - gibt es in Form eines Hooks: useReducer

können für State-Handling aber auch zusätzliches Framework einsetzen: im React-Bereich bsw. Redux-Framework - auch Redux verwendet Reducer-Konzept

Typischer Aufbau / Struktur eines React-Projektes:



npm modul Create React App

Modul erstellt Beispiel-App mit idealer Grundstruktur
Modul bringt Script mit, dass in der Lage ist, die App auch für Server ohne NodeJS auszuliefern
erstellt Beispiel-App: npx create-react-app my-app

App für Produktivbetrieb aufbereiten: npm run build
Erstellt die App für den Produktivbetrieb im Build-Ordner.
bündelt React korrekt und optimiert den Build für beste Leistung
Build wird minimiert und Dateinamen enthalten Hashes
App kann sofort bereitgestellt werden

react-router-dom - npm modul für Routing: npm install react-router-dom
Funktionsweise: https://reactrouter.com/web/example/basic

React Developer Tools

sehr gute Hilfe, um Fehler aufzuspüren
Installiere sie als Chrome Erweiterung aus dem Chrome Web Store
React Developer Tools erweitern Chrome Developer Tool um spezifische Funktionalitäten für React
React Inspector entspricht Standard-Inspektor, zeigt aber zusätzlich ReactKomponenten im DOM-Tree an – können bsw. props hier direkt ändern und Auswirkungen im Browser sehen